LOAN DOCUMENT

AD-A276 846

DTIC ACCESSION NUMBER

PHOTOGRAPH THIS SHEET

LEVEL

1

INVENTORY

WL-TR-94-1033

DOCUMENT IDENTIFICATION

Sep 93

DISTRIBUTION STATEMENT

ACCESSION FOR

NTIS GRA&I

DTIC TRAC

UNANNOUNCED

JUSTIFICATION

BY

DISTRIBUTION/

AVAILABILITY CODES

DISTRIBUTION	AVAILABILITY AND/OR SPECIAL	
A-1		

DISTRIBUTION STAMP

DTIC

MAR 10 1994

C

DATE ACCESSIONED

DATE RETURNED

94-07743

94 3 8 113

DATE RECEIVED IN DTIC

REGISTERED OR CERTIFIED NUMBER

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-FDAC

DTIC FORM 70A JUN 90

DOCUMENT PROCESSING SHEET

PREVIOUS EDITIONS MAY BE USED UNTIL STOCK IS EXHAUSTED

LOAN DOCUMENT

HANDLE WITH CARE

WL-TR-94-1033

A FRAMEWORK FOR DEVELOPING AND MANAGING REUSABLE AVIONICS SOFTWARE

DR. RAGHAVA G. GOWDA

SEPTEMBER 1993

FINAL REPORT FOR 05/01/93-09/01/93

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7409

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Charles P. Butterbrodt

Project Engineer
WL/AAAF-3

Robert L. Harris

Chief, WL/AAAF-3

Diana M. Morris

Chief, WL/AAAF

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WL/AAAF, WPAFB, OH 45433-6543 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

SEP 1993 FINAL 05/01/93--09/01/93

A FRAMEWORK FOR DEVELOPING AND MANAGING REUSABLE AVIONICS SOFTWARE

C
PE
PR 9994
TA 00
WU 00

DR. RAGHAVA G. GOWDA

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7409

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7409

WL-TR-94-1033

DEVELOPING REUSABLE SOFTWARE FOR AVIONICS SYSTEMS IS A CHALLENGE TO SOFTWARE ENGINEERS. THE TASK INVOLVES FORESEEING FUTURE APPLICATIONS, MODELING REAL-WORLD EVENTS, USING APPROPRIATE CASE TOOLS THROUGHOUT THE DEVELOPMENT LIFE CYCLE, AND PROVIDING FOR STORAGE AND RETRIEVAL OF REUSABLE SOFTWARE COMPONENTS, BRIEFLY DESCRIBES THE SOFTWARE PROCESS MATURITY MODEL AND THE INTEGRATION OF CASE TOOLS WITH THE PROCESS MATURITY MODEL. IT ALSO IDENTIFIES TOOLS/TECHNIQUES AND METHODOLOGIES FOR REAL-TIME SYSTEMS DEVELOPMENT, EXAMINES CRITICAL ISSUES IN MANAGING SOFTWARE PROJECTS, AND OFFERS THE MANAGEMENT A SET OF GUIDELINES TO INTRODUCE SOFTWARE ENGINEERING METHODOLOGIES.

UNCLASSIFIED UNCLASSIFIED UNCLASSIFIED UL

A Framework for Developing and Managing Reusable Avionics Software

Raghava G. Gowda, Ph.D.
Assistant Professor
Department of Computer Science

University of Dayton
300 College Park
Dayton, OH 45469

Final Report for:
Summer Faculty Research Program
Software Concepts Group (WL/AAF-3)
Avionics Directorate
Wright-Patterson Air Force Base
USAF Focal Point: Charles P. Satterthwaite

Sponsored by:
Air Force Office of Scientific Research
Bolling Air Force Base, Washington, D.C.

September 1993

A Framework for Developing and Managing Reusable Avionics Software

Raghava G. Gowda, Ph.D.
Assistant Professor
Department of Computer Science
University of Dayton

Abstract

Developing reusable software for avionics systems is a challenge to software engineers. The task involves foreseeing the future applications, modeling real-world events, using appropriate CASE tools throughout the development life cycle, and providing for storage and retrieval of reusable software components. This report presents a model for developing and managing reusable software components, briefly describes the software process maturity model and the integration of CASE tools with the process maturity model. It also identifies tools/ techniques and methodologies for real-time systems development, examines the critical issues in managing software projects, and offers the management a set of guidelines to introduce software engineering methodologies and CASE tools within the organization through a model project which may enforce standards for new projects.

A Framework for Developing and Managing Reusable Avionics Software

Raghava G. Gowda, Ph.D.

INTRODUCTION

It has been a great opportunity for me to work at the Wright Laboratory's Avionics Logistics Branch. During my stay, I had opportunities to do detailed literature survey on software reusability, go through some documentation on some of the projects, talk to consultants assessing software process maturity of the branch, interact with the software CASE tool vendors, talk to various project managers, and act as a consultant for one of the projects under development using object-oriented approach.

The initial objective of the summer research program was to study reusability issues in avionics software. The approach taken was to do :

A. A detailed library search in reusability and identify design criteria for reusable software
B. Gather design details about reuse efforts such as Reusable Ada Avionics Software Packages (RAASP) and Common Ada Missile Package (CAMP)
C. Summarize findings.

As it was not possible to get complete information on these projects, the research effort was directed to develop a framework for developing and controlling reusable software in the avionics domain. Based on my research, and participation in weekly review meetings of a simulation project, and my observations during my stay here, I would like to present a model for developing and managing any real-time software systems such as avionics systems. The model would be applicable for laboratories which initiate, fund, and control projects as well as to contractors who develop the systems.

Because the subject is broad, only a few aspects of it have been emphasized in this report. It is not possible to offer detailed discussions due to page constraints. After defining the reusability of software, a life cycle model is proposed which recognizes maintenance and further development as a part of the development process. This may be treated as a life cycle model of reusable software (or close to it). Then, the role of software maturity model and CASE tools in software development is discussed and suggestions are offered for integration of CASE tools acquisition and maturity model. A number of critical issues in managing software development tasks are outlined. Finally, suggestions are offered to develop a model project within an organization using appropriate methodologies and CASE tools. This will help an organization to attain higher levels in software process maturity model and offer insights for managing and integrating various projects. This experience may form a foundation for future reusable avionics software development tasks. The report consists of the following major topics:

1. Software Reusability
2. A Life Cycle Model for Reusable Software
3. Software Process Maturity Model for in-house management and external control.
4. CASE Tools
5. Software Process Maturity Model and Role of CASE Tools
6. Critical Issues in Managing Software Projects:
    - 6.1 Contents of Proposals and Deliverables
    - 6.2 Domain Analysis for Avionics and Assessment of Available Technology
    - 6.3 An Integrated View of all Projects
7. A Model For Developing and Managing Reusable Avionics Software
    - 7.1 A Model Project
    - 7.2 Software Development Processes, Methodologies, and Metrics
    - 7.3 Using CASE Tools in Projects
    - 7.4 Action Plan

1. SOFTWARE REUSABILITY

The reusability of software is the ultimate goal of any software development effort. The emphasis has shifted from project-specific systems analysis to domain analysis in order to incorporate flexibility in analysis and design so that the project-specific efforts can be reused in a broader domain. Developing reusable software for avionics systems is a challenge for software engineers. The task involves foreseeing future applications, modeling real-world events using appropriate tools, techniques, and methodologies for analysis and design of the system, and translating the specifications to software, verifying correctness of the software by testing, and providing for storage and retrieval of reusable software components. The software engineering discipline offers a number of methodologies, tools and techniques, and metrics to assist various phases of software development activities. The CASE (Computer Aided Software Engineering) technology integrates most of the tools and techniques.

Reusability has been defined differently by various authors. Kernighan [KER84] defines it as "...any way in which previously written software can be used for a new purpose or to avoid writing more software." Bott *et al.* [BOT86] give a more pragmatic definition of reusability: " a measure of the ease with which a component may be used in a variety of application contexts." In general, reusability of software can be interpreted from multiple view points. First, it can be seen as the use of existing software within a changing environment [SCH87]. Second, it can be viewed as the construction of new programs by composing such a program from software components [LYO86]. Third, it can be interpreted as the construction of programs by program transformation [CHE84].

Reuse can generally be classified as the reuse of ideas, vertical reuse, horizontal reuse, and total reuse [HAL87]. Publishing methods and techniques including algorithms leads to reuse of ideas by software professionals. Vertical reuse refers to reuse in a particular language, and horizontal reuse refers to the widely used

components in a particular environment, such as, sort utilities. Total reuse, however, is the use of complete packages after some customization.

The term reuse applies to the products developed throughout the development life cycle of software which includes requirements specifications, logical and physical design, code, and any information needed to create software. A Reuse Taxonomy by Prieto-Diaz [PRI93] shows six views of software reuse:

1. By-Substance defines the essence of the items to be reused. It consists of ideas, concepts, artifacts, components, procedures, and skills.
2. By-scope defines the form and extent of reuse. Such reuse can be classified as vertical and horizontal reuse.
3. By-mode defines how reuse is conducted. It may be planned, systematic or ad-hoc, opportunistic.
4. By-technique defines the approach used to implement reuse. It could be either compositional or generative.
5. By-intention defines how elements will be reused, as black-box (as is) or white-box (modified).
6. By-product defines what work products are reused. It includes source code, design, specifications, objects, text and architecture.

The broader concept of reuse (Basili, 1988) includes the 'use of everything associated with a software project including knowledge." The emphasis in industry has been on artifacts reuse such as Booch Ada Parts collection and the Generic Reusable Ada components for Engineers. Two of the often cited reusable projects in Avionics are Common Ada Missile Package project (CAMP) and Reusable Ada Avionics Software Packages (RAASP). The increased focus on software reusability is attributed to an overall realization of the potential benefits of not only reusing code, but also using all aspects of the development process. The documentation produced at various levels should serve as sources for maintenance and reusability. Reusability of software can not be a uniform process; it always has to be tailored for a particular domain.

1.1 ROLE OF USERS AND DEVELOPERS IN DEVELOPING REUSABLE SOFTWARE

Users of software can be classified as immediate users of the software for whom software was developed in the first place, and future users of the software. Immediate users are more concerned about the ability of the software in meeting requirements for the application, budget, and time factors. Their objectives, however, may not be the same as those of future users who tend to generalize software attributes, which in turn could lead to additional costs and delays in development efforts and enhancements. Immediate users may not favor domain analysis unless they realize the need for future developments and also its cost implications. Future users could play

a role similar to that of assembly line workers who assemble software components to produce a new product. The reuse of this same software, however, will vary depending on the ability of the user and the features of the software components.

The abilities of the users of software components are a combination of skill levels of the users, familiarity of the problem domain, and the time taken to retrieve required components from reuse repository. The tasks of the users will also be facilitated by the characteristics of the components themselves. These component characteristics are incorporated in the software by the software engineer and the development team.

It should be emphasized that the major players in software reusability are users and developers . A Reusable Components Repository is the common base through which they interact. Users are concerned with the ease of using the repository and developers are concerned with populating the repository and keeping track of engineering details.

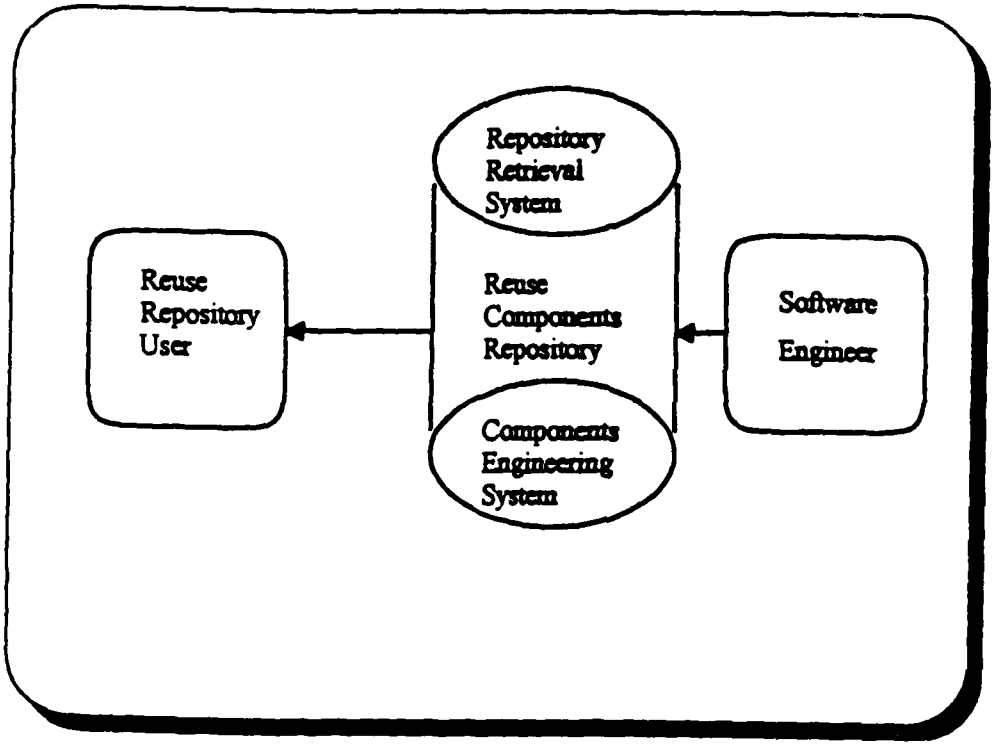

Figure 1. *Reusability Scenario*

The reusability scenario has two distinct features: a Repository Retrieval System and the Components Engineering System which consists of domain analysis, requirements analysis, design, metrics, and other configuration management details. Both the Repository Retrieval System and Components Engineering System have to be designed in collaboration with each other. A Repository Retrieval System is similar to a library retrieval system, but it has to consider graphical interfaces and capabilities to interface with other software repositories planned by DoD and commercial software vendors. Even though reusable efforts are evolving, standards for interface with repositories are not easily available. Therefore, retrieval systems need flexible designs.

Extracting reusable components from existing software is a complex activity and may not be cost effective in all cases. The basic information needed for reusable components from the users' perspective are:

a. Domain knowledge

b. Overview of reusable components

c. Details of each component which include:

Source code adhering to a particular style

Domain analysis

Requirements analysis of the system for which the component was designed

Logical Design

System (hardware/software) constraints

Testing / Quality Assurance reports

Unique features etc.

Other aspects.....

2. A LIFE CYCLE MODEL FOR REUSABLE SOFTWARE

Software Development Life Cycles offer a framework for software development. Though there is some concern about the life cycle concept itself amid its use, one has to admit their utility in identifying various tasks involved in the software development. Software professionals are well-versed with the following life cycle models:

1. Classical Software Development Life Cycle
2. Structured Software Development Life Cycle
3. Software Engineering Life Cycle
4. Information Engineering Life Cycle
5. Spiral Model of software development and enhancement

The life cycle models present different points of view for systems development. For example, the classical model assumes that the various activities are sequential. This model has been widely critiqued for its inability to

incorporate changing system needs. The *Structured Life Cycle* considers back-tracking and incorporating changes introduced at various phases. The *Software Engineering Life Cycle* closely follows the Classical and Structured life cycles and emphasizes walkthroughs, inspections, reuse, metrics, and deliverables. *Information Engineering Life Cycle* emphasizes defining data requirements of an enterprise first, and then the processes. It may be more appropriate for developing Management Information Systems projects. The *Spiral Life Cycle* deals with the prototype development environment, where neither the developer, nor the user have complete knowledge of the product to be developed. It considers risk analyses for different prototype developments. None of the above models treat software reuse or maintenance as a part of the models. Incorporating maintenance and reuse paradigm in the model forces the developer to consider a futuristic view of the system during development.

Henderson-Sellers and Edwards [HEND90] propose the **Fountain Model** (Figure 2) for the object-oriented life cycle which consists of the traditional life cycle phases and three additional phases of *Program Use, Maintenance,* and *Further Development* at the top of the Fountain. The model extends the Waterfall or Structured Life Cycle models to include reuse, maintenance, and further enhancements of software. The Fountain Model represents object-oriented software life cycle. It may also be treated as reusable software development life cycle, where *Maintenance*, and *Further Development* are some aspects of reuse efforts. We would like to add Domain Analysis prior to Requirements Analysis phase in the Figure 2 to emphasize the fact that reusability issues should be domain specific, because designing software for universal reusability may not be cost-effective and practical.

Domain analysis plays a significant role in the development of reusable software. Domain analysis refers to a detailed survey of the past efforts of an application area, current development tasks, and future needs of the problem domain. Domain analysis will allow the managers and developers to integrate past and current efforts, schedule for the optimum utilization of resources, and substantial savings in time and efforts in development, maintenance, and enhancements of systems.

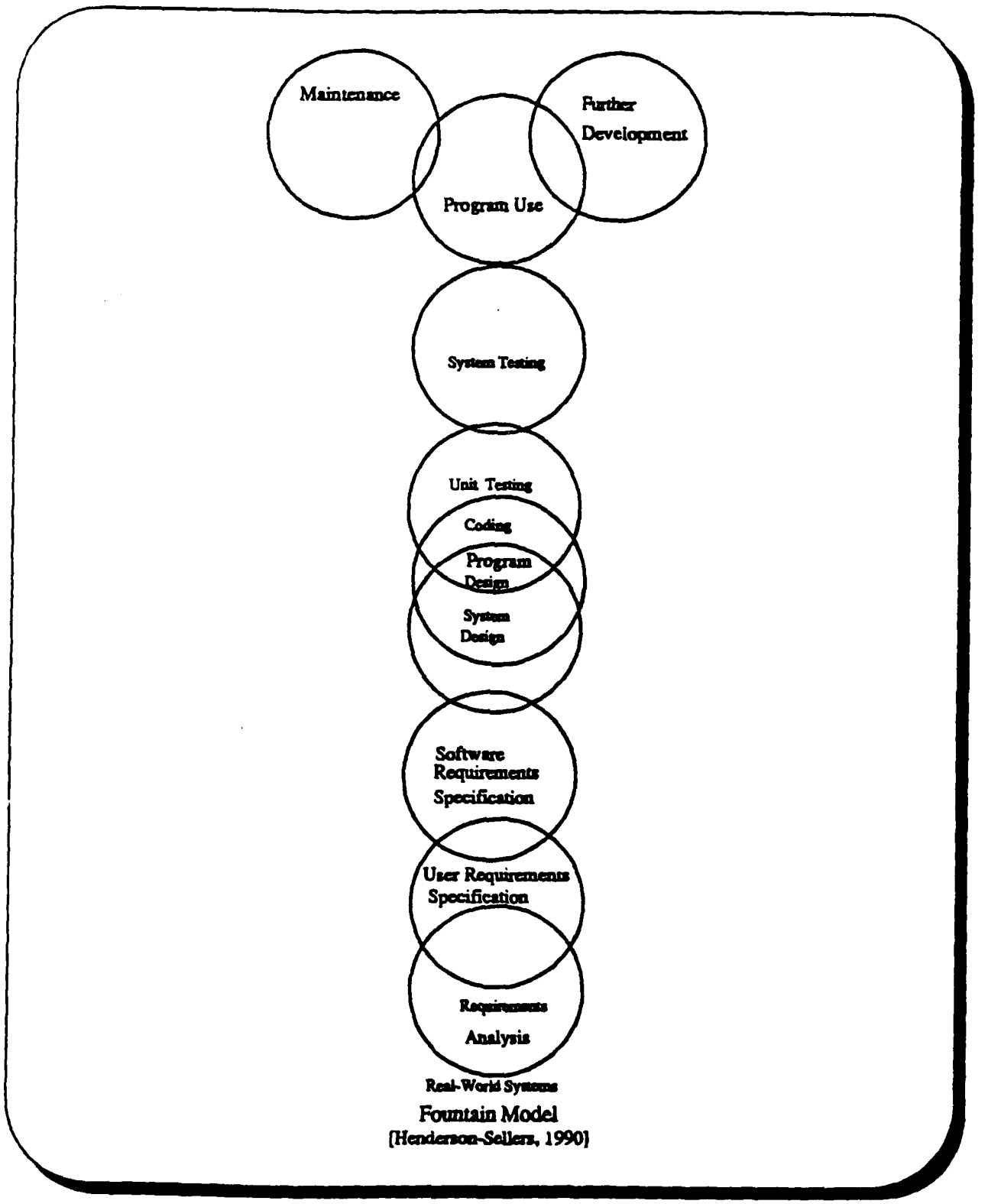

Figure 2. *Fountain Model*

3. SOFTWARE PROCESS MATURITY MODEL FOR IN-HOUSE MANAGEMENT AND EXTERNAL CONTROL

The basic elements of a management process are: planning, organizing, operating and control. For an organization such as Software Concepts Group at the Avionics Directorate of the Wright-Patterson Air Force Base, which forecasts technology for future avionics systems, translates the concepts to feasible projects, funds and monitors the projects, and diffuses the technology to its customers, and to the community at large, management issues become very complex. We can view management functions from two perspectives:

1. Internal control
2. Controlling projects developed by contractors.

The task of foreseeing the future technology is the most crucial one. As the projects are of diverse nature, it may not be possible to use the same set of tools/techniques or CASE tools in all the projects. However, a few guidelines could be offered to maintain uniformity among projects. It is necessary to:

a. Provide a common format for all projects;
b. Maintain control over all documentation and source codes;
c. Record contributions of individual projects with respect to contribution to knowledge, technology, products, etc. ;
d. Encourage use of software engineering principles and CASE tools wherever appropriate;
e. Integrate contributions of all the projects and practice them in the subsequent projects.

The Software Process Maturity Model [HUM89] would be equally applicable to Laboratories controlling projects as well as to contractors involved in software development. If the Laboratories have a high level of process maturity then they will be in a position to demand and control appropriate deliverables from their contractors more effectively. The software process is the entire set of tools, methods, and practices used to produce a software product.

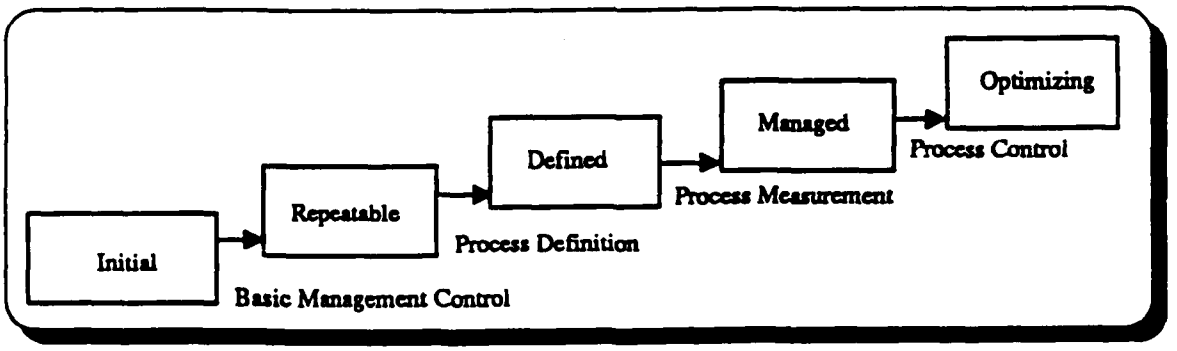

Figure 3. *Software Process Maturity Levels*

Control and improvement of tools, methods, and practices can lead the organization to higher maturity levels. Metrics or measurement of outputs can play a major role in translating the art of software development to the science of software development. CASE tools will also play a major role in the software process maturity model.

4. CASE TOOLS

This section outlines the specifics of the avionics software development process. The demands of avionics software development process are quite distinct from those of Management Information Systems or any other commercial applications. Most of the avionics applications are embedded real-time systems. The new applications are to be developed using Ada. Some of the tools/techniques and methodologies required to specify the requirements and design of the avionics systems are as follows:

A. Structured Analysis and Design Methodology

Tools/Techniques Used:

1. Data Flow Diagrams and its Real-Time extensions
2. Data Dictionary
3. Data Modeling and Entity Relationship diagrams
4. Decision Tables, Decision Trees, Action-Diagrams, Nassi-Shneiderman charts
5. State Transition Tables, State-Transition Diagrams
6. Finite-State Architecture
7. Petrinets
8. Structure Charts (in Structured Design Methodology)

B. Data Structure-Oriented Methodologies

1. Warnier/Orr Methodology
2. Jackson System Development (JSD)

C. Real-Time Design Methodologies

1. Design Method for Real-Time Systems (DARTS)
2. Structured Analysis and Design Technique (SADT)

D. Object-Oriented Analysis and Design Methodologies:

1. Bailin: Object-Oriented Requirements Specification
2. Coad and Yourdon: Object-Oriented Analysis and Object-Oriented Design
3. Shlaer and Mellor: Object-Oriented Analysis
4. Wasserman et al. Object-Oriented Structured Design (OOSD)
5. Booch: Object-Oriented Design
6. Wirfs-Brock et al. Responsibility Driven Design (RDD)

7. Rumbaugh et al. Object-Modeling Technique (OMT)
8. Embley et al. Object-Oriented Systems Analysis (OSA)

These methodologies can be implemented in most of the leading CASE tools. In general, the CASE tools can be classified from multiple view points such as upper CASE, lower CASE, and I-CASE (Integrated CASE Tools). Some of the unique components of the CASE Tool set are [BUR 89] summarized below:

a. Diagramming Tools
b. Syntax Verifiers
c. Prototyping Tools
d. Code Generators
e. Project Management and Methodology Support
f. Re-Engineering Tools
g. Central Repository

At present about 24% of the software development organizations use CASE tools. Most of the tools implement Structured Systems Analysis and Design methodologies, but they are not well integrated with code generation and testing tools. One of the major problems is the compatibility among CASE tools. Lack of industry standards makes it difficult to port systems from one set of tools to another. It also makes users dependent on a particular tool vendor. Organizations generally are hesitant to adopt this new technology because of high cost investments in these tools as well as the costs involved in training employees in the methodologies and tools. Moreover the management of an organization has high expectations about pay-off from these tools but employees are reluctant to experiment with the new technology. The opponents of CASE tools may be concerned about the following issues:

1. Training with methodologies
2. Ease of use of CASE Tools
3. Redundancies in documentation
4. Poor integration of various phases of life cycle
5. High costs of acquisition and maintenance
6. Probability and compatibility of the tools

These are some of the concerns. It should be kept in mind that CASE technology is still evolving and it might take a while to have an I-CASE in its real sense. If an organization ready to cope with the technology, it is the time to start now! We have to keep in mind that acquiring knowledge always comes incrementally. Training employees with software engineering methodologies is clearly the most crucial aspect. A careful evaluation of the tools that will be needed should be done prior to their acquisition. Consultations with users of some of the tools in similar projects seems to be the most desired method for evaluation of the tools. It would be an ideal approach,

however, to train project managers on a Model Project which would be of common interest. Various software engineering tools/techniques can be experimented in the Model Project. For example, an expert in the methodology may act as a moderator in the development process. In learning the development process education in software engineering disciplines would be more beneficial as compared to training with a specific CASE tool because it is not the tool which decides the success of any project but the modeling or problem solving approach which dictates how well the tool is used to develop the system.

Most CASE tools are capable of producing documents adhering to 2167A standards. However, it is the responsibility of the management to enforce uniformity in the contents of deliverables by giving specific directions regarding formats, tools/techniques, methods, and cross-referencing in the documentation. The ultimate objective is always to get an integrated product which is easy to operate, maintain, and reuse. This could be accomplished by enforcing standards with respect to the tools, techniques, and other process details.

5. SOFTWARE PROCESS MATURITY MODEL AND ROLE OF CASE TOOLS

Pfleeger [PFL91] suggests that "only when development process possesses sufficient structure and procedures does it make sense to incorporate certain kind of CASE tools in a development environment." He advocated the following CASE tools for the various process maturity levels.

	Level	Characteristics	Metric to use	CASE Tools
1.	Initial	Adhoc	Baseline	Tools that help to structure and control, estimate product size and effort
2.	Repeatable	Process dependent on individuals	Project	Tools for requirements specification, project management, and configuratiom management
3.	Defined	Process defined, institutionalized	Product	Tools to measure quality, complexity, to support design and coding, and to guide testing and integration
4.	Managed	Measured process (quantitative)	Process + Feedback	Project database, management system, simulation tools, reliability models, and impact analysis
5.	Optimizing	Imrovement fedback to process	Process + Feedback for changing process	Process programming and process simulation tools

Figure 4. *Process maturity levels related to CASE tools*

6. CRITICAL ISSUES IN MANAGING SOFTWARE PROJECTS

Issues involved in software project management are numerous and quite complex. Some of the key issues which have tremendous impact on deliverables, usability, and control of the projects are the following:

6.1 CONTENTS OF PROPOSALS AND DELIVERABLES

After going through a number of proposals, work plans, deliverables, etc. I realized the difficulties in managing the projects. Even with my adequate background in the software engineering area, I felt that I would be at the mercy of the contractor if I were a manager. The reason is that I may not have enough control over the projects. The DoD has all the controls in place for managing the projects through standards such as 2167A, but the documentation I read raised the following questions which could serve as guidelines for an organization:

1. What are the contributions of this project in terms of knowledge and technology?
2. Have similar projects been undertaken by the same contractor, by other agencies, or by other organizations in U.S. or abroad?
3. How well the contractor completed the "Related Work" section in the proposal?
4. Can we have full control over the deliverables?
5. Does the contractor make us depend on the particular contractor for related efforts in future by the nature of deliverables, or mode of information sharing?
6. How well the requirements specification is separated from design issues?
7. Has the continuity among various phases of systems development been demonstrated by the contractor?
8. How well the project is integrated with other projects in the Avionics Directorate?
9. How easy is it to follow documentation and other deliverables?
10. Do we have metrics to assess deliverables?

6.1.1 SOURCES OF CONFUSION

I strongly believe that one of the major sources of confusion can be in the proposal itself, in which the contractor fails to distinguish between the requirements of a system and the design of the system. Systems analysis or requirements analysis should emphasize only on "what" is expected of the system, and not "how" it is to be done. Unfortunately most of the proposals I read did not distinguish between analysis and design. This lack of distinction blurs the basic objectives of the project. Additionally, it is difficult to evaluate the approach as no alternatives are given. The proposal may be too detailed about a specific mode of implementation so that the reader is lost in the details or led to believe that it is the only way to do things. In such a scenario the manager has no option but to accept the tasks as outlined in the proposal. The manager will not have much control over the project as the proposal did not specify microscopic details. Only an expert in the particular area may raise

meaningful questions. The other alternative which I may be forced to adopt because of the nature of proposal, is to keep quiet and be satisfied with whatever deliverables are received from the contractor. To overcome these difficulties, I would ask the contractor to strictly follow the following guidelines:

1. Specify the system requirements without referring to any hardware or product details.
2. Discuss implementation details in the Design aspects.
3. Do not mix "What" and "How" aspects.
4. Describe contents of deliverables. Especially, identify what type of methodologies, CASE tools etc. would be used for analysis and design. Describe how documentation of one phase will lead to the successive phases.

6.1.2 TECHNOLOGY VS. APPLICATIONS

Investment in research projects related to technology development is potentially a high return area if it is conceived and managed very carefully. Otherwise investments may not produce substantial outputs. Every effo t should be made to diffuse the technology to the community at large and break the monopoly of a few vendors. In avionics domain, the main thrust is both in software development and technological advancement. In both the cases, outputs of the projects need to be carefully evaluated and controlled.

6.1.3 MONOPOLY OF SOFTWARE DEVELOPERS

In the 1960's Original Equipment Manufacturers (OEM) controlled the hardware industry because of their unique coding schemes. This approach made the user dependent on a particular vendor of hardware. Introduction of ASC II solved this problem to some extent. Now a similar scenario exists today in the software industry. Though there are standards such as 2167A for documentation of software, the contents of documentation vary because of different tools and techniques used in software development, and the lack of detailed standards and process control.

In the absence of matured processes for software development, a reasonable control on the project can be achieved by defining the *contents* of deliverables. To some extent the contents would define the process and enforce uniformity in managing projects. One of the aspects which need to be emphasized, however, is to separate logical requirements from implementation details. This is the key to extend the life of the system.

6.2 DOMAIN ANALYSIS FOR AVIONICS AND ASSESSMENT OF AVAILABLE TECHNOLOGY

Domain Analysis is the formulation of the common elements and structure of a domain of applications []. It is a way of understanding and describing the past and present in order to increase productivity in the future. It

should be an ongoing process. Domain Analysis in avionics refers to surveying the avionics applications and assesses the role of the current and past projects in the avionics domain applications. One of the objectives of domain analysis is to develop a framework for integrating all tasks in avionics domain. Some of the questions which may be useful in this context are:

a. What is the knowledge base for major air frames for example, (current and future fighters, bombers, helicopters, and trainers) regarding their current functions, missions, environments and future plans?

b. What are the goals and objectives of this group with respect to software and hardware under its control?

c. Do we know the details of all current projects, and deliverables within a suitable format?

d. Do the current projects meet goals and objectives? To what extent?

e. What applications are better for reusability, in terms of :

i) analysis

ii) design

iii) code

iv) algorithms

f. Are projects aimed to develop prototypes and concepts documenting the outputs in such a manner that they could be used/implemented by any independent team?

g. Which projects undertaken by DoD, ARPA, NSF, other agencies, and private organizations are similar to the current one? Describe them.

h. Is a format for domain analysis documents developed?

6.3 AN INTEGRATED VIEW OF ALL PROJECTS

Each project should have documentation on various phases. DoD Standard 2167A deals with the details and these should be followed. For the purposes of managing projects, the Avionics Directorate may develop a document giving an integrated view of the projects under its control. A yearly report may be compiled which will consist of the following essential aspects of all the projects and how they are integrated:

1. Domain Analysis
2. Contributions of the Project
3. How these contributions are integrated with other Avionics projects or any other efforts?

The major emphasis of this yearly report should be on concepts on technology advances and on aspects which could be used elsewhere. Implementation details are available at the project level and as such should not be part of this report.

7. A MODEL FOR DEVELOPING AND MANAGING REUSABLE AVIONICS SOFTWARE

Based on the past discussions, a Model for Developing and Managing Reusable Avionics Software is offered in Figure 5.

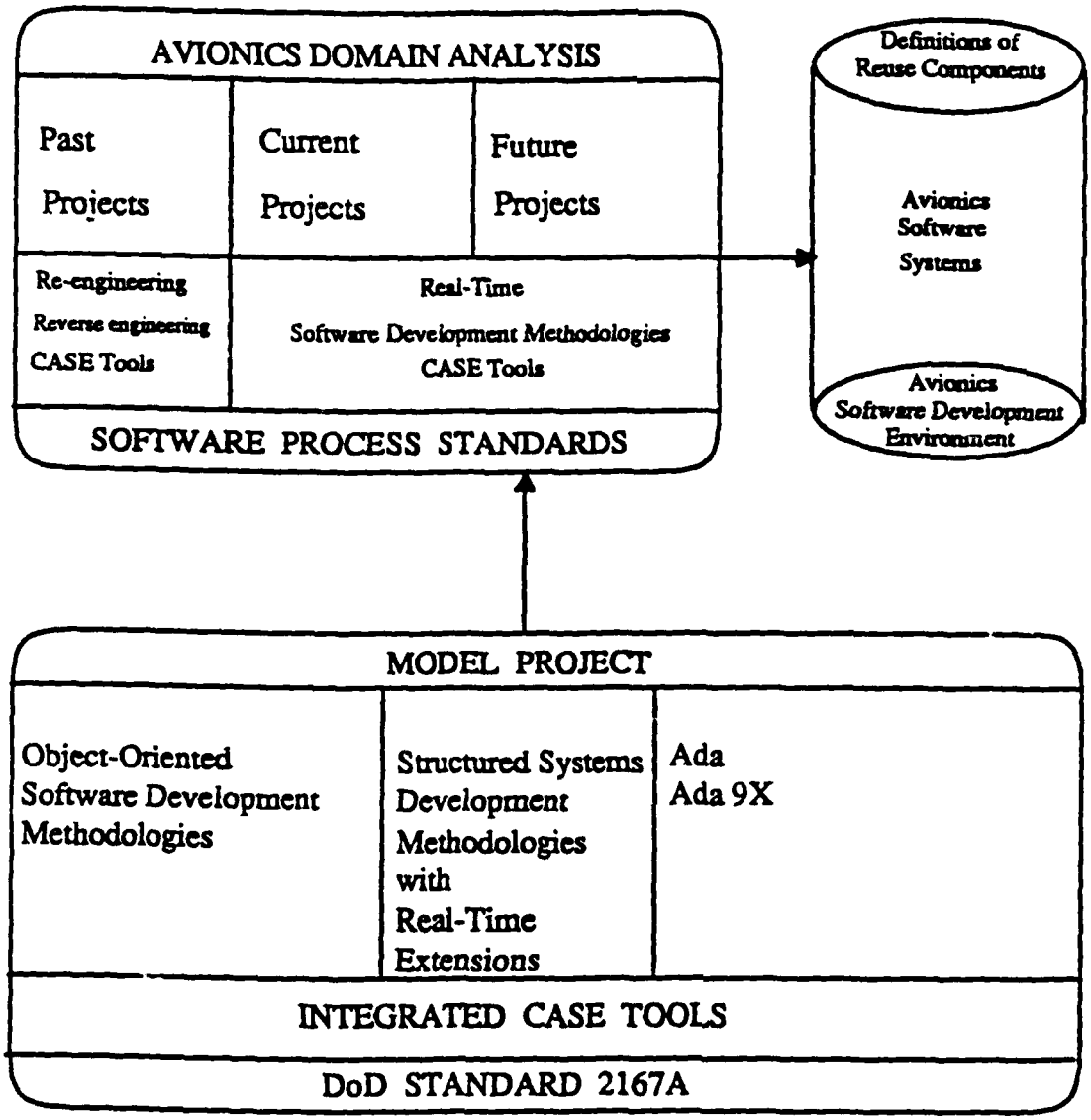

Figure 5. *A Model For Developing and Managing Reusable Avionics Software*

7.1 A MODEL PROJECT

To standardize the project formats and to appreciate technical details, it will be useful to work on a Model Project with close collaborations with a developer or consultant. It may be useful to re-engineer an existing project which will have high reusability because the existing project will have all technical details. If, however, a new project is selected the emphasis will shift to new technical details and exploration aspects. Therefore, a new project may be restrictive in exploring all tools/techniques, methodologies, and other issues. The difficulty in re-engineering an existing project would depend on the availability of documents, style of coding, and the language used.

The Model Project should be developed using appropriate methodologies, CASE tools, and documentation should adhere to standard 2167A . It should serve as a model for use of methodologies and documentation. This experience may be valuable for in-house development and in managing software development projects by contractors.

7.2 SOFTWARE DEVELOPMENT PROCESS, METHODOLOGIES, AND METRICS

Avionics software systems have a number of characteristics which make them unique compared to the traditional data processing application. In general, they are real-time systems which need to activate as a response to external events. Response time is critical for such systems. Some of them may be embedded where software is integrated with hardware components. Correctness of software, and reliability are of utmost concern to users of the software. Therefore, testing of software is a very elaborate task in its development. The system needs to be maintainable at the earliest possible time. On-board testing and maintenance efforts would minimize the machine-down-time. Most of the tools and techniques identified earlier would also be applicable to avionics software development.

Object-oriented methodologies are very well-suited for developing reusable software. The concepts of abstraction, information-hiding and polymorphism are practiced in these methodologies. Though there are a number of OOA, OOD, OOA and OOD methodologies, they are not in a matured stage such as Structured Methodologies. The Object Modeling Technique by Rumbaugh et al. [RUM91] seems to be one of the most appropriate methodologies for avionics system development as it offers Object Model, Dynamic Model, and Functional Model facilities for requirements specification of a system. It is a nice blend of structured and object-oriented concepts. The methodology may need further refinements to provide continuity among the three models and to represent collaboration between various subsystems and objects. A CASE Tool called OMTool™ implements the methodology. However, it is to be kept in mind that at present OMTool™ does not implement the Dynamic Model and Function Models of Rumbaugh's methodology. Representation of collaborations between

objects and integration of different models are the major issues regarding this methodology. The Model Project may use Object-Oriented and Structured methodologies in analysis and design phases and Ada in implementation phase. Ada 9X incorporates object-oriented programming features.

7.3 USING CASE TOOLS IN PROJECTS

At present most of the CASE tools used are in analysis and design areas. The analysis and design methodologies are not well integrated. CASE tools related to project management, code generation, and testing should also be used wherever applicable. This would integrate all the phases of life cycle. A recent report from Institute for Defense Analysis has identified that there are more than 600 testing tools available but not widely used. The CASE technology is heading towards I-CASE or Integrated CASE. Management may introduce these tools progressively in projects. As discussed earlier, acquiring different tools may be linked to different levels of process maturity.

7.4 ACTION PLAN

The action-plan may emphasize standardizing processes for a project. A participative mode of implementing a change in an organization may face less resistance. The manager may do the following for instance:

1. State the need for a Domain Analysis for Avionics to identify the future technological needs.
2. Show how the existing projects meet the needs. Emphasis should be on specifics. Global words should be avoided.
3. Identify goals for the next five years.
4. Draw an implementation plan which covers:
    a. A suggested format for summary and detailed internal reports for projects
    b. Details of deliverables from contractors
    c. Software Engineering tools/techniques to be used
    d. Identify the need for a Model Project
    e. Invite suggestions from Project Engineers
    g. Incorporate suggestions and implement the plan.

One of the difficult tasks would be to bring projects with multiple dimensions under a common thread of control. This has to be done by studying individual projects and then comparing their similarities and differences. The model project may promote uniformity in the processes i.e. in tools, methods and practices which would lead the organization to a higher level of process maturity and force the contractors to adhere to higher standards.

REFERENCES

[BAK90] Baker,T.P. Software Reuse in Real-Time Environments. Report submitted for U.S. Army HQ CECOM, Center for Software Engineering, October 9, 1990.

[BAR93] Barnes, J. Introducing Ada 9x. Ada 9x Project Report, Office of the Under Secretary of Defense for Acquisition, Washington DC, 20301, 1993.

[BUR89] Burkhard, Donald L. Implementing CASE Tools. ASM Journal of Systems Management, May 20, 1989.

[BOT86] Bott, M. F., A. Elliott and R.J. Gautier. Ada Reuse Guidelines. - Report, ECLIPSE/REUSE/DST/ADA_GUIDE/RP, Alvey ECLIPSE Project Deliverable D36, February 1986 Software Sciences Ltd.

[CAT91] Cattel, R.G.G. Object Data Management. Addison-Wesley, Reading, Mass., 1991.

[CHA92] Champeaux, Dennis de., and Penelope Faure. A Comparative Study of Object-Oriented Analysis Methods. JOOP Journal of Object-Oriented Programming, March/April 1992.

[CHE84] Cheatham, T. Reusability through Program Transformation. IEEE Transactions on Software Engineering, V 10 (5), pp. 589-594, September, 1984.

[HAL87] Hall, Patrick A. Software Components and reuse - getting more out of your code. In Will Tracz (Ed.) Tutorial: Software Reuse: Emerging Technology, The Computer Society of the IEEE, 1988.

[HEN90] Henderson-Sellers, B. and Edwards, J.M. The Object-Oriented Systems Life Cycle. Communications of the ACM, September, 1990.

[HOL90] Holmgren, Brian W. Software Reusability: A study of why software reuse has not developed into a viable practice in the Department of Defense. Thesis submitted to Air Force Institute of Technology, December 20, 1990.

[KER84] Kernighan, B.W. The UNIX System and Software Reusability. In IEEE Transactions on Software Engineering, pp. 513-518, 1984.

[HUM89] Humphrey, Watts S. Managing the Software Process. Addison-Wesley, 1989.

[INT91] Intermetrics. Draft Ada 9X Mapping Document, Volumes I and II. Mapping Specification, Ada 9X Project Report, August 1991.

[LYO86] Lyons, T. G.L. and Nissen, J.C.D. (Eds.) Selecting an Ada Environment. Cambridge University Press, 1986.

[PFL 91] Pfleeger, S.L. Process maturity as framework for CASE tool selection. Information and Software Technology, November 1991.

[PRI93] Prieto-Di'az, Ruben. Status Report: Software Reusability, May 1993, pp. 61-66.

[RUM91] Rumbaugh, James., Blaha, Michael., Premerlani, William., Eddy, Frederick., and Lorensen, Wiiliam. Object-Oriented Modeling and Design. Prentice Hall, 1991.

[SCH87] Schneidewind, N.F. Introduction to the Special Section on Software Maintenance. IEEE Transactions on Software Engineering, V13(3), pp. 303-310, March 1987.